Révisions de Java et remarques diverses

Alexandre Honorat

Mai 2019

1 Programmation : principes généraux et Java

La programmation informatique consiste à exprimer des opérations qu'un ou plusieurs ordinateurs doit exécuter, dans un langage compris à la fois par les humains et par les ordinateurs. Java est un de ces langages, particulièrement populaire et l'un des plus utilisés dans le monde [1].

1.1 Qu'est-ce qu'un programme ?

Un programme informatique est une liste de traitements à effectuer sur des données. Par analogie, un programme ressemble à une recette de cuisine où les traitements sont les instructions données au cuisinier et où les données sont les ingrédients utilisés. Plus précisément Java fait partie des langages orientés objets, où les traitements et les données peuvent être stockés au même endroit : dans une classe. Une instance de classe est justement appelée un objet. Par analogie, on peut dire qu'une classe est le plan d'une voiture, tandis que l'objet est la voiture elle-même. La programmation en Java est dite impérative, c'est-à-dire que tous les traitements ont la possibilité de modifier les données stockées (mais il n'est pas obligatoire de le faire) et on parle d'effets secondaires [2].

Les données informatiques sont très généralement structurées en types. Les types les plus courants abstraient les nombres entiers et décimaux : `short`/`int`/`long` et `float`/`double` par exemple [3]. En Java, les nombres sont des types dits primitifs, de même que les booléens `boolean`, les caractères `char` et les octets `byte`, et ils ont tous une précision limitée (2^8 valeurs pour les `byte` et 2^{16} valeurs pour les `short`). Les types primitifs s'opposent aux classes qui servent à définir des structures bien plus complexes, à partir d'autres classes ou des types primitifs. De ce fait, une classe contient toujours un ou plusieurs constructeurs dont le rôle est d'initialiser les données stockées dans l'objet à une valeur par défaut ou passée un argument. En résumé, les classes structurent les données, en même temps qu'elles fournissent des traitements de ces données. Les traitements fournis par une classe sont appelées méthodes.

1. C. f. https://insights.stackoverflow.com/survey/2019#technology
2. La dénomination exacte est « side effects » en anglais, qui est dans 99% des cas mal traduit par « effets de bord » alors que cela n'a rien à voir avec des quelconques « bords ».
3. Les nombres purements rationels ou transcendants, à représentation infinie, ne sont supportés que par les langages de mathématiques formelles.

1.2 Comment représenter un programme ?

Avant d'écrire un programme, il faut avoir une idée de ce qu'il va faire. Les représentations mathématiques des calculs à effecture sont les algorithmes [4]. Un programme Java est généralement constitué de plusieurs classes, contenant chacune plusieurs méthodes implantant des algorithmes. Une seule de ces classes contient une méthode spécifique, appelée `main`, étant l'unique point d'entrée, ou autrement dit de départ, du programme. C'est l'instruction numéro 1. de la recette de cuisine.

Comme de nombreuses classes peuvent constituer un programme, on les représente généralement schématiquement au sein d'un graphe Unified Modeling Language (UML). Dans un tel graphe UML, chacune des classes est schématisée par un rectangle, qui comporte son nom, les noms des données qu'elles comportent (appelées attributs), et enfin les noms des méthodes. Les flèches du graphe servent à indiquer différentes notions telles que la possession d'attributs ayant le type de la classe pointée (avec un losange du côté de la classe contenante, et le nom de l'attribut au-dessus de la flèche), ou l'héritage des méthodes et des attributs (flèches normales et en pointillées). Avant d'écrire un programme, on réfléchit donc d'abord à l'organisation des classes schématisée en UML, et aux algorithmes à implanter. Au final, le programme est une implantation [5] du graphe UML et des algorithmes.

1.3 Comment utiliser un programme ?

Une fois écrit le programme doit être compilé ou interprété afin d'être exécuté par l'ordinateur. Un programme Java est d'abord compilé dans un langage proche du langage machine sans en être dépendant : il s'agit du bytecode contenu dans les fichiers `.class` générés pour chacune des classes `.java`. Le bytecode généré est ainsi multi-plateforme et peut être exécuté sur n'importe quel ordinateur disposant du Java Runtime Environment (JRE). Le JRE interprète le bytecode en un langage compréhensible par le modèle du processeur installé sur l'ordinateur et l'exécute simultanément. La compilation des classes se fait via la commande `javac` du Java Development Kit (JDK) tandis que l'interprétation et exécution du bytecode se fait via la commande `java` du JRE.

Il est important de noter que le processeur d'un ordinateur ne comprend que des informations très limitée, principalement : charger un nombre depuis une case mémoire, stocker un nombre dans une case mémoire, additionner ou soustraire, multiplier ou diviser, et enfin tester l'égalité ou l'inégalité de deux nombres. Les cases mémoires sont identifiées par des adresses, elles-même des nombres stockés sur 64 bits (i. e. 8 octets, comme un `long`) sur les processeurs modernes. Le rôle du compilateur est donc de convertir toutes les instructions du programme dans les instructions très limitées sur processeur, et plus particulièrement il identifie le nom des variables avec les cases mémoires dans lesquelles elles vont être stockées. C'est là la principale différence entre les types primitifs et les classes : alors que les types primitifs sont stockés directement avec le nom de la variable, les objets de classes sont stockés dans des cases mémoires séparés du nom de la variable qui y fait référence. Une variable d'objet ne stocke donc qu'une adresse appelée référence ; l'adresse est déterminée à l'exécution, lors de l'appel du constructeur via `new`. Par ailleurs, le compilateur vérifie les types de toutes les variables, afin que les structures de données soient toutes utilisées correctement (sauf dans certains cas spécifiques, notamment lors du cast d'une classe mère vers une classe fille, c. f. section 3 sur l'héritage).

2 Classes Java

Pour écrire une classe, il est nécessaire de respecter la structure syntaxique (aussi appelée grammaire) de Java : les éléments doivent être écrits en respectant un certain ordre et les noms de mot-clés réservés. Si une classe ne respecte pas la syntaxe de Java, elle ne peut pas être compilée.

4. Vraissemblablement, le premier algorithme connu est celui d'Euclide, pour calculer le plus grand diviseur commun de deux nombres entiers. Toutefois, le mot algorithme est apparu bien plus tard et correspond au nom du scientifique perse Al-Khwarizmi

5. La plupart des traductions utilisent le mot « implémentation », qui est un anglicissme, au lieu du correct « implantation ».

```java
package exemple.insa; //les packages servent à regrouper certaines classes entre elles

import java.util.ArrayList; //il faut importer les classes que l'on utilise
import java.util.List;
import ...;

public class MaClasse extends ClasseMere {

  //partagée entre toutes les instances, non modifiable
  static final String NOM_CLASSE = "MaClasse";

  static int nbInstances = 0; //partagée entre toutes les instances, modifiable

  //propre à chaque instance (instance = objet)
  List<Integer> maListe; // on déclare avec le type de l'interface


  public MaClasse() { // constructeur, initialise les attributs
    super(); //appel du constructeur de ClasseMere
    maListe = new ArrayListListe<Integer>();
    nbInstances++;
  }

  public void methodeA(int a) { // méthode ne renvoyant rien
    maListe.add(a); //this n'est pas utile s'il n'y a pas d'ambiguité de nom
  }

  public boolean methodeB() { // ici la méthode doit renvoyer un booléen
    return maListe.isEmpty(); // il y a au moins un return si la méthode n'est pas void
  }

  public void methodeC() throws Exception { // exemple d'exception
    int[] tab = new int[10]; // on crée un tableau de 10 entiers
    try {
      tab[-1] = 0; // problème ! la numérotation commence à 0
    } catch (ArrayIndexOutOfBoundsException e) {
      throw new Exception(e); // voir la section sur les exceptions
    }
  }

  // un et un seul main, dont le prototype est toujours le même :
  public static void main(String[] args) {
    System.out.println("Ça commence ici.");
    MaClasse a = new MaClasse();
  }

}
```

Listing 1: Syntaxe générale d'une classe Java.

Parmi les mot-clés les plus importants on trouve :
— ceux modifiant la visibilité d'un élément : `public` généralement, voir table 1 ;
— ceux modifiant le type de classe : `class`, `abstract class`, ou `interface` ;
— ceux modifiant un attribut : `static` pour le partager et `final` pour lui attribuer une valeur non modifiable au moment de la déclaration ou dans le constructeur au plus tard (si non `static`) ;
— toutes les structures de contrôle : `for`, `while`, `if` et `else`, c. f. listing 2.

```
//length n'existe que pour les tableaux, pour les listes c'est la méthode size()
for (int i = 0; i < tab.length; i++) {...}

//contenant doit être de type Collection<>, par exemple List<Integer>
for (Contenu c: contenant) {...}

// boucle tant que la formule booélenne est évaluée à true
while ( 1 < 2 || (!bool && NOM_CLASSE.equals("MaClasse"))) {...}

// boucle au moins une fois, puis tant que la formule booléenne est vraie
do {... } while (...)
```

Listing 2: Syntaxe générale des boucles.

	class	package	subclass	other
(+) public	Y	Y	Y	Y
(#) protected	Y	Y	Y	-
(~) —	Y	Y	-	-
(-) private	Y	-	-	-

TABLE 1 – Visibilité des éléments en fonction de l'endroit d'accès (et symbôle UML).

Le plus important est de ne pas oublier de construire les objets de classe que l'on utilise : il doit y avoir un `new` pour chaque classe instanciée. Si l'on utilise une méthode ou un attribut d'une variable de type non primitif qui n'a jamais été instanciée, la référence stockée est l'adresse spéciale `null` et cela génère alors une `NullPointerException`.

Par ailleurs il faut un et un seul point d'entrée du programme étant la méthode `main`, attention à son prototype d'ailleurs, à écrirer dans n'importe quelle classe (généralement la classe correspondant à la structure de donnée la plus structurante). Le prototype d'une méthode est constitué de son type de retour, de son nom, du type et de l'ordre de ses arguments (mais pas leurs noms), et des éventuelles exceptions qu'elle génére. Pour faire simple, le prototype d'une méthode est la première ligne du code qui lui correspond.

Toutes les classes commencent obligatoirement par une lettre majuscule, par opposition aux noms de variables, attributs ou méthodes. Il existe une classe correspondant à chaque type primitif, par exemple `Integer` pour `int`. Enfin il faut faire attention à l'indentation afin d'améliorer la lisibilité du bode : à chaque fois que l'on entre dans un bloc de code délimité par des accolades, on doit écrire un peu plus à droite (avec toujours le même espacement, par exemple 4 espaces ou 1 carreau).

3 Polymorphisme en Java

Le polymorphisme en Java peut provenir de la surcharge de méthode (méthode avec le même nom qu'une autre, mais un prototype différent), de l'héritage, et de la généricité. La surcharge ne pose pas de difficulté particulière, l'héritage et la généricité sont en revanche un peu plus complexes.

3.1 Héritage

L'héritage permet de simplifier l'écriture d'un programme en partageant les attributs et méthodes d'une classe avec toutes celles en héritant (sauf si ces attributs et méthodes sont précédées de `private` ou aucun mot-clé). On hérite d'une classe via le mot-clé `extends`, et on ne peut hériter que d'une seule classe au total. À l'inverse il est possible de définir des interfaces ne décrivant que les prototypes de méthodes (sans le code) ; alors, une même classe peut implanter plusieurs interfaces à la fois via le mot-clé `implements`, en même temps qu'elle hérite ou non d'une autre classe. L'héritage multiple est interdit pour éviter les conflits de choix d'une méthode si plusieurs méthodes ont le même prototype mais des codes différents ; ce problème ne se produit pas avec les interfaces puisque seuls les prototypes sont déclarés dans l'interface. Lorsqu'une classe implante une interface, il faut fournir le code des méthodes de l'interface. Enfin, à mi-chemin se trouvent les classes abstraites, qui ne fournissent que le prototype de certaines méthodes (déclarées `abstract`). Les classes abstraites et les interfaces ne peuvent pas être instanciées puisqu'elle ne contiennent pas tout le code des méthode abstraites qu'elles déclarent. De plus les interfaces n'ont pas de constructeurs ni d'attributs, seulement des méthodes `public`. Lorsqu'une classe normale hérite d'une classe abstraite, il faut obligatoirement écrire dans la classe fille le code des méthodes abstraites de la classe mère. Cela n'est en revanche pas obligatoire si une classe abstraite hérite d'une autre classe abstraite.

Les relations possibles d'héritage sont résumées dans la table 2.

⟶	class	absract class	interface
class	E	E	I+
abstract class	E	E	I+
interface	—	—	E+

TABLE 2 – Relation d'héritage possible (+ si multiple), `extends` (E) ou `implements` (I) entre les différents types de classes. À lire par ligne.

En Java, toutes les classes héritent de la classe Object. Même si l'héritage est unique, une classe hérite d'une classe qui hérité d'une ... qui ell-même hérite de Object. Deux méthodes sont particulièrement importantes dans la classe Object : `+toString(): String` et `+equals(Object o): boolean`. Il est le plus souvent nécessaire de les redéfinir dans les classes filles. Comme pour toutes les méthodes redéfinies, c'est le code de la classe fille qui est appelé. Pour accéder au code de la classe mère il faut précéder l'appel par le mot-clé `super` (qui permet aussi d'appeler le constructeur de la classe mère).

```
public boolean equals(Object o) {
  if (o instanceof MaClasse) {
    MaClasse mc = (MaClasse) o; // cast vers un sous-type
    return mc.maListe.equals(this.maListe);
  }
  return false;
}
```

Listing 3: Redéfinition classique de `equals`.

`equals` doit toujours être privilégié à `==` pour comparer deux objets, car `equals` permet de comparer les attributs si elle a été redéfinie tandis que `==` ne compare que les adresses des références ! Autrement dit `==` permet de retrouver l'objet original tandis que `equals` permet de retrouver un objet différent mais ayant des propriétés similaires. En revanche, pour les types primitifs, `==` compare bien les valeurs des nombres. Par ailleurs, notez que la division entière de deux nombres entiers renvoie la partie entière inférieure, il faut penser à faire un cast d'un des deux nombre en `float` ou `double` si l'on veut obtenir un résultat décimal.

3.2 Généricité

La généricité permet d'écrire le même code, adapté à plusieurs types de variables différents. Le type est choisi lorsque l'on instancie l'objet, et en attendant il est identifié par une lettre majuscule, c. f. listing 4. Le type mit entre chevron ne peut être qu'une classe, les types primitifs ne sont pas acceptés. Dans une liste par exemple (l'interface `List<E>` est générique), tous les éléments du type spécifié ou dérivés de celui-ci peuvent être ajoutés à la liste.

```
public class Generique<T,U,V> {
  public T methode(U arg1, V arg2) {
    ...
  }

  public static void main(String[] args) {
    Generique<MaClasse, Double, Float> = new Generique<MaClasse, Double, Float>();
    MaClasse a = methode(1.0, 2.0F);
  }
}
```

Listing 4: Redéfinition classique de `equals`.

4 Exceptions en Java

Les exceptions permettent de gérer les erreurs afin, quand cela est possible, de ne pas faire planter le programme. Elles permettent dans tous les cas de mieux identifier la source de l'erreur. On distingue deux types d'exceptions : les `Exception` qui doivent obligatoirement être gérées et les `RuntimeException` qui elles, peuvent ne pas être gérées, et arrêtent alors le programme.

Les exceptions sont gérées par un bloc `try/catch` comme dans listing 1. Il peut y avoir plusieurs `catch`, placés dans l'ordre inverse des relations d'héritages, c'est-à-dire les classes filles en premier. Attention, c'est bizarre et pas très cohérent, mais il se trouve que `RuntimeException` hérite de `Exception`, quand bien même elles sont traitées différemment par le compilateur. À la fin du bloc il est possible de rajouter un unique `finally` afin d'effectuer des opérations telles que la fermeture d'un fichier, avant de propager l'exception à la méthode appelante. Il est possible de ne pas gérer une `Exception` uniquement à condition de la propager à la méthode appelante via le mot-clé `throws` dans le prototype (à différencier de `throw` qui sert à lancer une exception). Si le `main` lui-même contient `throws` dans son prototype, alors l'exception arrêtera le programme.

5 Bibliothèques usuelles Java

Les bibliothèques Java sont des ensembles de classes déjà codées, fournies par les développeurs de Java. Elles concernent plusieurs usages courants tels que la lecture de fichier ou le stockage d'éléments. Le travail d'informaticien consiste surtout à réutiliser les bibliothèques existantes qui font ce que l'on souhaite, et les informaticiens n'en développent de nouvelles que si nécessaire. La documentation est alors essentielle pour comprendre du code qui a été fait par d'autres. L'exemple le plus utile de bibliothèque concerne les fonctions mathématiques : `Math.abs`, `Math.sqrt` et `Math.pow` notamment.

5.1 Comparateurs

Les comparateurs permettent de comparer deux objets en renvoyant 0 s'ils sont considérés égaux, -1 si le premier objet à comparer est inférieur au second, et $+1$ si c'est l'inverse. Deux interfaces génériques

permettent de faire cela :

— `Comparator<T>` a pour seule méthode `public int compare(T o1, T o2)` et nécessite la création d'une classe l'implantant ;

— `Comparable<T>` a pour seule méthode `public int compareTo(T o)` et est implantée directement dans la classe `T`, comparant donc l'objet courant à `o`.

```
public class MaClass implements Comparable<MaClasse> {
  public int attribute;
  ...

  int compareTo(MaClasse arg0) {
    return Integer.compare(this.attribute, arg0.attribute);
  }
}
```

Listing 5: Exemple classique de comparable.

5.2 Collections d'éléments

L'interface `Collection` précise des méthodes générales d'accès et de stockage d'élément au sein d'une même collection. Ci-dessous les principales interfaces/implantations, toutes génériques, à connaître :

— `Collection<>/-` : interface la plus générale dont les suivantes héritent/implantent ;

— `Set<>/HashSet<>` : ensemble au sens mathématique ;

— `List<>/ArrayList<>` : liste ordonnée ;

— `Map<>/HashMap<>` : annuaire, i.e. collections de couple clé-valeur avec clés uniques.

Il existe des versions triées de certaines collections : `TreeSet<>` et `TreeMap<>` stockent par ordre croissant (des clés pour les `Map`). Par ailleurs, l'interface `Collections` (attention au s) dispose d'une méthode `sort(List<T> list)` permettant de trier une liste (en place), ainsi que d'une surcharge pour utiliser un comparateur spécifique, `sort(List<T> list, Comparator<? super T> c)` [6].

5.3 Itérateurs

Les itérateurs permettent de parcourir tous les éléments d'une structure, et notamment de toute `Collection`. Attention, les itérateurs sont à usage unique : une fois parcourue en entier, il est nécessaire d'instancier un nouvel itérateur afin de parcourir de nouveau une collection. Les itérateurs découlent de deux interfaces :

— `Iterable<>` : implantée par toutes les `Collection`, permet d'obtenir un itérateur ;

— `Iterator<>` : permet de parcourir l'ensemble des éléments.

```
Iterator<Integer> it = maListe.iterator();
while (it.hasNext()) {
  int a = it.next();
  // traitements utilisant a, mais ne pouvant pas modifier sa valeur dans maListe
}
```

Listing 6: Exemple classique d'itérateur.

6. `<? super T>` signifie qu'il doit s'agit de `T` ou de n'importe quelle classe située au-dessus de `T` du point de vue de l'héritage. Les classes dérivées de `T` ne sont pas acceptées. C'est un point technique qu'il n'est pas nécessaire de comprendre.

5.4 Gestion des fichiers

Voir le TD associé. Principales classes et méthodes à connaître :
— `String.split(';')` : pour séparer une chaîne de caractères en sous-chaînes délimitées par `';'` dans ce cas, renvoie un tableau ;
— `Integer.parseInt(String entier)` : pour transformer une chaîne de caractère en `int`, idem pour les autres types primitifs ;
— `FileReader/FileWriter` : pour lire/écrire dans un fichier, attention à la gestion des exceptions, et on n'oublie pas de fermer le fichier après utilisation via `close()`, dans un bloc `finally` ;
— `InputStreamReader` : pour lire depuis une entrée utilisateur, notamment `System.in`, aussi à fermer après utilisation ;
— `BufferedReader` : pour lire une entrée ou un fichier ligne par ligne, aussi à fermer après utilisation.

5.5 Interface graphique JavaFX

Concernant JavaFX, les principaux points à retenir hors détails techniques sont :
— Modèle Vue Contrôleur (MVC) : sépare la réalisation graphique des données stockées ;
— conteneurs graphiques : les conteneurs détérminent la façon d'afficher les éléments graphiques, un conteneur peut contenir d'autres conteneurs, le plus imbriqué contient un Widget (bouton ou autre).

6 Complexité

La complexité d'un algorithme s'exprime par rapport à la taille du problème : très souvent le nombre d'éléments dans un tableau ou le nombre de nœuds dans un graphe. La complexité concerne deux aspects de l'exécution : le temps d'exécution et l'espace mémoire. Pour le temps d'exécution, on compte alors le nombre d'opérations effectuées par l'algorithme, généralement ce sont les multiplications, les additions ou les comparaisons de valeurs. Enfin, comme la complexité est parfois dépendante des données elles-même (à taille de problème égale), il est souvent calculé une complexité en moyenne, au pire, et au mieux.

Les notations utilisées pour la complexité sont plus ou moins celles de Landau (les définitions peuvent varier), pour des problèmes de taille n :
— majoration asymptotique $O(g(n))$ (complexité au pire) ;
— minoration asymptotique $\Omega(g(n))$ (complexité au mieux) ;
— ancadrement asymptotique $\Theta(g(n))$ (complexité dans tous les cas, différent de en moyenne !).

Définitions :
— $f(n) \in O(g(n))$ ssi $\exists k \in \mathbb{R}^+$, $\exists n_0 \in \mathbb{N}^*$, $\forall n > n_0$, $|f(n)| \leq k|g(n)|$
— $f(n) \in \Omega(g(n))$ ssi $\exists k \in \mathbb{R}^+$, $\exists n_0 \in \mathbb{N}^*$, $\forall n > n_0$, $k|g(n)| \leq |f(n)|$
— $f(n) \in \Theta(g(n))$ ssi $\exists k_1, k_2 \in \mathbb{R}^+ \times \mathbb{R}^+$, $\exists n_0 \in \mathbb{N}^*$, $\forall n > n_0$, $k_1|g(n)| \leq |f(n)| \leq k_2|g(n)|$

Attention, d'après les définitions ici utilisées, $f(n) \in O(g(n)) \not\Leftrightarrow g(n) \in \Omega(f(n))$, en effet le facteur k n'est pas placé du même côté de l'inégalité.

En pratique, on omet de préciser le nom de la fonction de complexité et l'on écrit plus simplement :
— $O(1)$: complexité constante (comme une opération arithmétique) ;
— $O(\log n)$: complexité logarithmique (recherche dichotomique dans un ensemble trié) ;
— $O(n)$: complexité linéaire (typiquement une boucle `for` sur un tableau) ;
— $O(n \log n)$: complexité linéarithmique (certains tris) ;
— $O(n^2)$: complexité quadratique (typiquement deux boucles `for` imbriquées) ;
— $O(2^n)$: complexité exponentielle (e. g. tours de Hanoï).

Lors du calcul de la complexité, si plusieurs chemins sont possibles dans le programme (via des `if/else`), il faut prendre le minimum des deux sous-complexités pour Ω et le maximum pour O. Lorsque le programme est récursif, c'est-à-dire que la méthode s'appelle elle-même, alors la fonction de complexité est également formulable récursivement (le cas d'initialisation est généralement égal à une constante). Attention la complexité de l'algorithme n'a rien à voir avec la fonction calculée : par exemple la complexité des algorithmes

calculant le produit factoriel ou la puissance sont linéaires. Cela est généralement rendu possible grâce à la *programmation dynamique*, dont le principe est de sauvegarder les résultats de la même fonction pour des tailles de problème plus petites (notamment Fibonacci). On parle aussi de mémoïsation.

7 Tris

Un tri consiste à ordonner les éléments d'une collection. Seules les relations d'ordre total sont considérées : c'est-à-dire que tous les éléments à trier doivent être comparable deux à deux [7]. Un tri est réalisé *en place* si les éléments triés sont triés et stockés dans la collection de départ (ce qui est possible pour une liste ou un tableau, mais pas pour un ensemble puisqu'il n'y aurait alors pas de notion d'ordre). Un tri est *stable* si les éléments égaux restent dans le même ordre que dans la collection de départ (et si celle-ci a une notion d'ordre bien sûr).

Les tris les plus simples sont le tri bulle et le tri par insertion, tous deux de complexité quadratique, stables, et en place. Le tri bulle s'explique simplement par le fait de faire remonter les éléments les plus grands vers la droite du tableau ; en revanche dans le tri par insertion, les éléments sont insérés suivant leur valeur dans une sous-partie triée du tableau, qui grandit jusqu'à être le tableau complètement trié. Les tris efficaces reposent quant à eux sur le paradigme « diviser pour régner ». Ce paradigme s'emploie en informatique pour diviser un problème en plusieurs sous-problèmes de taille inférieure (typiquement, par 2), afin de calculer la solution plus aisément. Les tris fusion et rapide reposent sur le paradigme « diviser pour régner » : le premier découpe récursivement le tableau initial en deux sous-tableaux (jusqu'à atteindre la taille 1) qui seront ensuite fusionnés deux par deux et maintenus triés [8] tandis que le second découpe récursivement le tableau initial en deux sous-tableaux dont tous les éléments du premier sont inférieurs à ceux du second. Ainsi le tri fusion découpe puis trie en fusionnant, tandis que le tri rapide découpe en triant par rapport à un élément pivot. L'étape de fusion n'est en fait même pas nécessaire dans le tri rapide puisque le tri s'effectue en place, et donc les sous-tableaux triés sont déjà côte-à-côte. Pour plus de détails sur les algorithmes de tris, voir le TD associé.

La table 3 résume les propriétés des différents tris. Notez que de meilleurs tris peuvent exister lorsque des informations supplémentaires sont connues : par exemple si les bornes minimum et maximum sont connues, un tableau peut être trié avec une complexité linéaire en comptant simplement le nombre d'occurence de chaque valeur entre les deux bornes, comme vu en TD.

Tri	complexité moyenne	pire des cas	stable	en place
Bulle	$O(n^2)$	$O(n^2)$	oui	oui
Insertion	$O(n^2)$	$O(n^2)$	oui	oui
Fusion	$O(n \log n)$	$O(n \log n)$	oui	non
Rapide	$O(n \log n)$	$O(n^2)$	non	oui

TABLE 3 – Principale caractéristiques des tris usuels.

8 Graphes

Les graphes sont présents partout en informatique et mathématiques. Les graphes sont constitués de nœuds, aussi appelés sommets, reliés entre eux par des arcs dans le cas dirigés et des arêtes dans le cas non dirigé. Les arcs et arêtes représentent donc une relation binaire non transitive et non réflexive entre les nœuds, cette relation est symétrique dans le cas non dirigé. Dans un graphe non dirigé il est ainsi possible d'emprunter une arête dans n'importe quel des deux sens. Les principaux problèmes liés aux graphes sont

7. Pour les ordres partiels, on parle de treillis, ou lattice en anglais.
8. L'algorithme permettant de fusionner deux tableaux déjà triés est linéaire en temps et en espace. Par ailleurs, remarquez qu'un tableau à un seul élément, est évidemment déjà trié.

l'établissement d'un chemin entre deux nœuds et la détection de cycles (un cycle est un chemin dont le premier nœud emprunté est également le dernier).

Formellement, on écrit généralement que $G = (V, E)$ avec G le graphe, V l'ensemble des nœuds, et E l'ensemble des arêtes ou arcs ; dans les graphes non pondérés $E \subseteq V \times V$, et dans les graphes pondérés $E = w : V \times V \longmapsto \mathbb{R}$ avec w la fonction de pondération. Les graphes peuvent être représentés de deux manières : par matrice d'adjacence (utilisées pour les calculs algébriques) ou par liste d'adjacence (utilisées pour les parcours dans les graphes). Une matrice d'adjacence est carrée de taille $\#V$, elle correspond donc à la valuation de w (et est symétrique dans le cas non dirigé). Une liste d'adjacence est de taille $\#V$ et est en fait un tableau de listes : pour chaque nœud $u \in V$, en index du tableau, une liste associée stocke les nœuds $\{v \in V | (u, v) \in E\}$.

Le parcours des graphes se fait via deux algorithmes principaux : Breadth-First Search (BFS) est un parcours en largeur tandis que Depth-First Search (DFS) est un parcours en profondeur. Concrètement BFS part d'un nœud puis visite tous les nœuds immédiatement connectés à celui-ci (on parle de voisins, ou d'enfants), puis tous les nœuds connectés aux voisins du nœud de départ, etc. DFS en revanche visite un nœud puis le premier enfant de celui-ci puis le premier enfant de ce dernier, etc, et visite les autres enfants d'un nœud uniquement lorsque tous les enfants du premier l'ont déjà été. Pour savoir quels nœuds ont été visités, on les *marque* (avec un booléen par exemple). Ce marquage permet notamment de détecter les cycles : lorsque l'on atteint un nœud qui a déjà été marqué, c'est que l'on ferme un cycle. Dans les implantations récursives de BFS et DFS vues en TD, la seule différence entre elles est l'ordre d'ajout des enfants dans la liste des prochains nœuds à visiter : à la fin pour BFS (la liste est alors une file d'attente, ou queue en anglais) et au début pour DFS (la liste est alors une pile, ou stack en anglais). Au début de l'algorithme, la liste des prochains nœuds à visiter ne contient que le nœud de départ. Une fois un nœud visité, il est enlevé de la liste et marqué, puis tous ses enfants sont ajoutés dans la liste. L'algorithme termine lorsque la liste est vide. Dans le cas pondéré avec w à valeurs dans $\mathbb{R}^+$ uniquement, l'algorithme de Dijkstra permet également de calculer le plus court chemin entre le nœud de départ et n'importe quel autre nœud avec un algorithme similaire, où la liste des prochains nœuds à visiter est maintenue triée.

Dans le cas dirigé et lorsque le graphe ne possède pas de cycle, on parle de Directed Acyclic Graph (DAG). Il est alors possible de calculer un ordre topologique $o : V \longmapsto \mathbb{N}$, c'est à dire une numérotation des nœuds qui garantit $(u, v) \in E \implies o(u) < o(v)$, et par fermeture transitive, que si un chemin existe entre u et v, alors $o(u) < o(v)$. Attention, plusieurs ordres topologiques peuvent exister pour un même DAG.

9 Bref historique de l'informatique

L'informatique est indissociable de deux autres disciplines : les mathématiques et l'électronique. Alors que le premier algorithme connu est vraissemblablement celui d'Euclide durant l'antiquité, pour calculer le plus grand diviseur commun de deux entiers, il a fallu attendre 2000 ans avant que l'algorithmique devienne une discipline à part entière. Le mot algorithme ne signifie d'ailleurs rien en soi, il provient du nom du mathématicien perse Al-Khwarizmi, ayant vécu au IX^{me} siècle.

L'idée de pouvoir effectuer des calculs automatiquement (et plus précisément, *mécaniquement*) a été concrétisée par Pascal au $XVII^{me}$ siècle, avec sa machine à calculer nommée « Pascaline », uniquement pour l'arithmétique entière (il était concurrencé par Leibniz). Deux siècles plus tard, Charles Babbage veut créer une machine (mécanique toujours) à calculer *analytique* : c'est-à-dire non limitée aux expressions arithmétiques. Il ne parvient pas à finir sa machine, mais dans le même temps il est aidé par Ada Lovelace, qui apprécie beaucoup les mathématiques. Ada Lovelace est la fille du poète Lord Byron, et donc un peu riche et très cultivée. Elle aide financièrement C. Babbage, et elle s'intéresse à un autre problème lié à la machine analytique : comment exprimer les calculs. Ce faisant, elle crée ... le premier algorithme pour une machine, sans que la machine ne soit encore construite. Pour la construire, Babbage s'est inspiré de la dernière innovation de l'époque, dans l'industrie du textile : le Métier Jacquard. Le métier (à tisser) Jacquard introduit en fait le principe même de programmation : pour tisser les motifs voulus, des cartes perforées à des endroits spécifiques actionnent mécaniquement la bonne alternance des fils lors du passage de la navette. C'est l'ancêtre de la programmation ; les cartes perforées en informatique contiennent les instructions et,

également les données (sur les mêmes cartes ou séparémment). Toutefois la machine analytique n'est achevée complètement qu'après la mort d'Ada Lovelace, Charles Babbage, ou Jacquard. Elle ne sera pas vraiment utilisée en l'état, mais inspire la création de plusieurs entreprises qui se concentrent sur des calculs plus spécifiques. L'entreprise la plus connue et ancienne dans le domaine étant IBM (créée en 1911).

On est au début du XX^{me} siècle, on a des machines capables de calculer à partir de cartes perforées, mais pas encore capables de tout calculer. Du point de vue mathématique, on se demande alors ce qui est calculable et ce qui ne l'est pas, problème associé à la décidabilité (ce qui est prouvable et ce qui ne l'est pas), étudiée notamment par Gödel. Inspiré par les travaux de Gödel et d'autres (Alonzo Church notamment), Alan Turing trouve la solution, et mieux que ça, il montre les limites de sa solution [9] . Premièrement il conçoit la machine (théorique) dite « machine de Turing » : il s'agit d'un ruban contenant des symbôles qui sont lus un par un par la machine, et la machine déplace le ruban ou écrit dessus grâce à une table associant chaque symbôle lu au déplacement ou à l'écriture à effectuer. Cette table, c'est le programme. Le ruban, ce sont les données. C'est mieux que les cartes perforées du métier Jacquard, qui elles, ne sont que lues, et que dans un sens. Mais surtout Turing sait ce que l'on ne peut pas faire avec sa machine : on ne peut pas créer de machine décidant à coup sûr si un programme va terminer ou non (une machine simulant une machine donc). Turing fonde donc l'informatique : il formalise la notion de programme.

Pendant la seconde guerre mondiale, l'informatique devient un avantage stratégique grâce aux travaux de Turing, qui aident à la fois à chiffrer les communications alliées et à décoder certains messages ennemis, chiffrés avec la machine Enigma. L'Allemagne disposait déjà de machines à calculer, créées notamment par Konrad Suze. Dans le même temps, les machines à calculer se sont mises à l'électricité, à la fois pour stocker les informations (les bits valant 0 ou 1), et pour traiter ces informations, mais toujours uniquement par des biais électro-mécaniques (des aimants notamment). Les programmes sont toujours sur des cartes perforées. À cette époque, la machine à calculer la plus généraliste est le Harvard Mark I (pesant plusieurs tonnes), créé par Howard Aiken pour IBM. Sur cette machine ont notamment travaillé deux autres pionniers de l'informatique : John von Neumann et Grace Hopper.

Ce n'est qu'en 1945 que le premier véritable ordinateur, car universel au sens de Turing (c'est à dire qu'il peut se comporter comme une machine de Turing), est créé : c'est l'ENIAC. L'ENIAC, américain, utilise le dernier cri technologique : les tubes à vides, il n'est donc plus électro-mécanique, mais bien électronique. Les tubes à vides n'ont alors été utilisés que par très peu de machines : notamment le Colossus britannique. Alors que tous les scientifiques précités ou presque sont des mathématiciens, c'est maintenant qu'intervient l'électronique. L'utilisation des tubes à vides, ancêtres des transistors, réduit un peu le poids et la place que prennent les ordinateurs. Mais la révolution arrive en 1947, avec le premier transistor moderne, théorisé 30 ans auparavant par Julius Lilienfeld. Les trois créateurs du transistor moderne sont récompsés par un prix Nobel en 1956 ; l'un d'eux, John Bardeen, recevra plus tard un deuxième prix Nobel pour d'autres travaux. C'est à partir de là que tout s'enchaîne : les transistors sont bien plus petits et moins énergivores que les tubes à vides. Les ordinateurs peuvent alors faire plus d'opérations et diminuer en taille et en poids. Par ailleurs il devient possible d'enregistrer les programmes et les données sur des bandes magnétiques, ce qui facilite encore plus la miniaturisation des ordinateurs. Cela se poursuit notamment jusqu'au premier alunissage, dont le navigateur de bord est contrôlé par un ordinateur de quelques kilos, programmé par l'équipe de Margaret Hamilton (pionnière du génie logiciel). À cette époque, toujours pas de micro-processeur cependant. Il faut attendre 1972, et le premier prototype de Federico Faggin pour Intel. De nouveau tout s'enchaîne ... l'ordinateur va devenir une machine pouvant être possédée par des particuliers. La physique a permis d'autres innovations majeures concernant principalement la manière dont les données sont stockées. Les lasers ont permis le stockage sur des CDs par exemple. Par ailleurs, l'effet tunnel, mis en évidence par Leo Esaki, est utilisé pour piéger les électrons (stockant donc un bit) dans la mémoire flash, comme dans les clés USB par exemple.

Suite à toutes ces innovations, l'informatique se développe en tant que discipline à part entière. Les innovations remarquables concernent alors autant les outils dédiés à la programmation, tels que le langage `C` et le système d'exploitation `Unix` créé par Dennis Ritchie et Ken Thompson, que les outils liés à la communication, tels qu'Internet (dont un des contributeurs majeurs est Time Berners-Lee), que les algorithmes théoriques

9. Parmi les autres personnalités célèbres dans le domaine de la logique mathématique et de la calculabilité on peut citer : George Boole, Emil Leon Post, Rózsa Péter et Stephen Cole Kleene.

(comme ceux de Donald Knuth et Leslie Lamport). Knuth a par ailleurs inventé le langage de typographie TEX, ensuite amélioré par Lamport pour donner la version moderne LATEX. Ce document, de même que la plupart des livres achetés en librairie, est écrit en LATEX par exemple.

Trop de noms sont à citer parmi les informaticiens célèbres. Certains ont été récompensés par l'un des deux prix les plus prestigieux du domaine [10] : le Prix Turing et le Prix Gödel. Toutefois l'attribution de ces prix est un peu biaisée. Le premier biais concerne le genre : aucune femme n'a été primée par le prix Turing de sa création jusqu'en 2006, soit pendant quarante ans. Pourtant, Grace Hopper par exemple aurait largement mérité d'y être [11]. Le deuxième biais concerne la nationalité des récipiendaires, en grande majorité américains pour le prix Turing. Pourant, le français Gilles Kahn ou l'allemand Carl Petri n'ont pas été récompensés pour leurs travaux fondateurs. Le troisième biais concerne les domaines récompensés, majoritairement proches des mathématiques, ce qui conduit à ne pas récompenser le concepteur du premier micro-processeur Federico Faggin. En France, les principaux laboratoires publics de recherche en informatique sont les unités sous tutelle du CNRS, ainsi que l'INRIA et le CEA.

10 Informatique et éthique (au sens large)

L'informatique a été au cœur d'une révolution technologique, qui n'est pas encore terminée. Toutefois cette révolution s'accompagne de son lot de bonnes et mauvaises pratiques. Voici quelques pistes pour comprendre et éviter de reproduire certaines des mauvaises pratiques ...

10.1 Un rapport ambigu à l'humanité

L'informatique a contribué au progrès technique essentiellement par les communications mondiales rendues possibles avec Internet, ainsi que par les innombrables calculs et autres simulations de systèmes mathématiques, physiques, chimiques ou économiques [12]. Du côté social, Internet (et Wikipédia notamment) a eu une influence considérable (et majoritairement positive) sur l'éducation et la transmission du savoir. Diverses initiatives offrent des cours accessibles gratuitement, comme Artips, une récente initiative du CNRS, celle du Collège de France, ou FUN, une plus large regroupant des universités, écoles et centres de recherches français. Des projets collaboratifs ont également permis de surpasser en qualité les services proposés par des entreprises, comme OpenStreetMap ; et maintenant les états ouvrent leurs données afin de mieux contrôler et coordonner l'action publique. Voilà pour le côté positif.

Maintenant le côté négatif. L'instigateur de l'iPhone, Jean-Marie Hullot [13], a par exemple, lors d'une conférence, essuyé de nombreuses questions négatives sur l'addiction que provoquerait l'iPhone (ou plutôt les applications qui y sont installées) sur certains enfants. Et il est vrai que les applications mobiles, souvent gratuites, ne sont généralement pas les plus respectueuses de la vie privée. Quand on vous offre un service, c'est que vous êtes le produit. Les dérives vont du marketing agressif à l'espionnage (plus ou moins consenti) des employeurs, par exemple envers leurs employées pour obtenir des informations sur leur grossesse. Les techniques utilisées sont assez classiques, et la récente loi sur la protection des données qui oblige à paramétrer les `cookies` ne va pas changer grand chose au pistage publicitaire : l'unicité de votre configuration de navigateur Internet suffit : `https://amiunique.org/` !

Mais voyons pire. L'informatique aime automatiser et rendre « intelligents » plein de programmes. Mais pour créer de tels programmes, il faut connaître une multitude de données et les saisir informatiquement. Rien de plus facile, il suffit d'acheter de la ressource humaine à prix coûtant, c'est ce qui est proposé par Amazon Mechanical Turk. Comment les pseudo « intelligences artificielles » sont-elles entraînées pour reconnaître des

10. À ce jour, seule une personne a été récipiendaire des deux prix. Il s'agit de la chercheuse Shafi Goldwasser.
11. La place des femmes en informatique, et plus spécifiquement au sein des contributeurs à Internet, est l'objet du livre Broad Band, The Untold Story of the Women Who Made the Internet, *Claire L. Evans*, Portfolio Penguin (en anglais, ISBN : 978-0-7352-1175-9). Ce livre ne considère par trop les avancées scientifiques cependant.
12. Une des premières simulations économiques a été réalisée par le Club de Rome en 1970, grâce à un programme informatique nommé World3. Les résultats étaient pour le moins inquiétants sur l'avenir de la planète ...
13. Hé non, ce n'est pas Steve Jobs ! Mais pour la petite histoire, c'est bien Steve Jobs qui a eu l'idée de ne pas mettre de clavier physique et a imposé à ses ingénieurs la principale innovation de l'iPhone : le plein écran tactile.

images ? En partie sur le dos de travailleurs du clic qui classent des images à la main afin de pouvoir former les ensembles d'images étiquetées. Et encore, ceux-là sont payés. Pour pouvoir s'y authentifier et soi-disant afin de renforcer la sécurité en vérifiant qu'il ne s'agit pas d'un robot, certains sites sur Internet demandent de sélectionner les images correspondant à tel objet, et utilisent donc les images ainsi étiquetées. Ce n'est pas le seul domaine ou les sites profitent de l'utilisateur, cela a également été le cas pour le minage du Bitcoin, une crypto-monnaie très éthique assurément (et très écologique de surcroît). Le blanchiment d'argent tout le monde sait que c'est très éthique, encore plus quand les internautes du monde entier y participent.

Du côté des états, il n'y a pas que les programmes civils qui poussent à financer l'informatique, notamment au CEA en France. Rappelons d'abord qu'Internet a été créé à la suite d'ARPANET, un projet financé pour l'armée américaine. Les télécommunications sont traditionnellement un axe de recherche militaire (souvenez-vous du rôle de Turing pendant la deuxième guerre mondiale), mais un autre axe est par la suite devenu primordial : la simulation des phénomènes physiques. D'une part afin de simuler les réactions nucléaires pour fabriquer des bombes, d'autre part afin de simuler les réponses aux ondes électro-magnétiques pour fabriquer des sous-marins, bâteaux, ou avions furtifs. C'est le cas d'un des plus puissants supercalculateurs du monde, TERA 1000, administré par le CEA. Actuellement, les télécommunications sont de nouveau l'enjeu principal, et les « progrès » réalisés sont imporants : presque n'importe quelle application de télécommunication peut être piratée, comme WhatsApp récemment [14]. En France, c'est l'ANSSI, en partie sous commandement militaire, qui se charge de surveiller le traffic et d'éviter les tentatives d'espionnage ou de déstabilisation des réseaux de télécommunications.

Enfin, une dernière dérive de l'informatique est la croyance qu'ont certains en le fait qu'elle doit augmenter, voire remplacer l'être humain. C'est le courant du transhumanisme, que presque tous les philosoppes français considèrent comme dangereux [15]. Par exemple certains transhumanistes croient que l'intelligence artificielle, et notamment les réseaux de neuronnes doivent à terme remplacer le cerveau humain. Mais c'est être bien optimiste quant à l'état d'avancement des réseaux de neuronnes qui fonctionnent bien pour classifier des images et relativement bien pour traduire des phrases, et ... c'est tout. Par ailleurs, il faut préciser que les réseaux de neuronnes actuels sont créés par des humains, et pour des humains. En France, le sujet est étudié par exemple par Jean-Gabriel Ganascia, qui est (malheureusement ?) relativement optimiste et confiant sur le futur de l'intelligence artificielle. Larry Page et Sergey Brin, les deux co-fondateurs de Google, sont transhumanistes. Par pitié, ne devenez pas comme eux, et faites plutôt de la philosophie.

10.2 Des programmes parfois mortels

Les erreurs des programmes informatiques, ou « bugs » (bogues en français), peuvent parfois coûter très cher voire tuer. L'un des exemples les plus connus est l' accident de la fusée Ariane 5, qui a explosé lors de son vol inaugural, causant plusieurs centaines de millions d'euro de pertes, et écornant la notoriété (depuis rétablie) des fusées Ariane. Et encore, ici il ne s'agit que d'argent : dans les années 2000, les régulateurs de vitesse des voitures Toyota ont causé une centaine de morts car l'accélérateur se bloquait en même temps que le frein était plus ou moins désactivé. Toyota a commencé par incriminer les conducteurs eux-mêmes [16], puis à du se résoudre à faire expertiser le code informatique et le matériel du système de freinage et de régulation. Une première inspection de la NASA n'a pas trouvé de faille, mais n'a pas non plus prouvé l'absence de faille [17]. Ce n'est qu'en 2013 qu'un autre expert est nommé par la justice américaine, Philip Koopman, et trouve l'origine du problème, qui est bien liée à un code mal écrit par Toyota. Ces conclusions sont présentées ici. Suite à ces accidents, et à bien d'autres malheureusement, les indutriels n'ont pas forcément pris la mesure des risques associés à un code non certifié, excepté dans l'aéronautique et le médical.

Mais quelles sont les origines de tels disfonctionnements et comment s'en prémunir ? Concernant les origines, il s'agit de mauvauses pratiques de gestion du personnel (quand on vous demande quelque chose d'impossible, le résultat n'est évidemment pas bon), et surtout de mauvaises pratiques de programmation.

14. L'application de communication la plus sécurisée actuellement semble être Signal.
15. Malheureusement je ne me souviens plus du nom du documentaire où plusieurs philosophes étaient interrogés à ce sujet. Toutefois récemment, Alain Badiou a publié une tribune où il fait allusion aux « impasses du transhumanisme ».
16. Ils étaient morts, c'était facile de les incriminer !
17. À propos des preuves en sciences, lire Popper, et plus généralement s'intéresser à l'épistémologie.

Bien coder est le sujet du génie logiciel : il faut que le code soit documenté, maintenable, et si possible modulaire et évolutif. Si on vous ennuie avec les commentaires, c'est pour que celui qui utilise votre code vous comprenne et ne fasse pas n'importe quoi avec [18]. Maintenant comment s'en prémunir ? On peut avoir un code propre mais non fonctionnel. Pour le certifier, on peut utiliser les *méthodes formelles*. Les méthodes formelles prouvent mathématiquement qu'un programme informatique se comporte comme il le devrait : il respecte des spécifications [19]. Plusieurs techniques existent, comme le model checking ou l'interprétation abstraite, d'ailleurs inventée par deux français : Patrick et Radhia Cousot. La France est relativement en pointe dans ce domaine, avec également le franco-grec Joseph Sifakis, récipiendaire du prix Turing et chercheur dans le domaine du model checking. On peut aussi citer Gérard Berry (contributeur aux langages synchrones utilisés pour la conception de systèmes aéronautiques) et Xavier Leroy (concepteur du premier compilateur formellement certifié CompCert), tous deux professeurs au Collège de France.

Petit aparté sur le sujet : la mode est maintenant d'utiliser l'intelligence artificielle et notamment les réseaux de neuronnes un peu partout. De par la nature même de ces réseaux, on ne sait pas certifier pour l'instant des programmes les utilisant. Donc si dans le futur proche vous montez dans une voiture autonome fonctionnant grâce à des réseaux de neuronnes, descendez-en de suite. Et vu la propension des constructeurs de voitures à tricher (DieselGate) et à incriminer les conducteurs, descendez vraiment le plus vite possible.

10.3 Propriété intellectuelle et écologie

L'informatique n'est pas écologique, et ce pour au moins trois raisons : 1) les composants électroniques utilisent des matériaux rares, 2) ces composants consomment et gaspillent de l'énergie, 3) le droit de la propriété intellectuelle. Les deux premiers points sont plutôt d'ordre matériel et en tant qu'informaticien, on ne peut pas y faire grand chose (sauf à utiliser l'approximate computing). Les processeurs actuels sont gravés en 14 nm (c'est la largeur des fils reliant les transistors), et sont donc très proches de la limite théorique de 7 nm à partir de laquelle les effets quantiques provoquent trop d'erreurs de résultats, car cela ne correspond qu'à quelques dizaines d'atomes de large. La finesse de gravure permet de limiter la déperdition d'énergie en chaleur par effet Joule. Toutefois la solution n'est peut-être pas seulement de limiter cette perte, mais aussi de la réutiliser pour du chauffage par exemple, technique malheureusement très très peu utilisée.

Concernant le droit de la propriété intellectuelle, le problème est le suivant : le code informatique est hautement réutilisable, sauf si sa réutilisation est rendue impossible, par certains droits de propriétés ou par l'offuscation (« obfuscation » en anglais). Cela a une conséquence directe : des milliers d'informaticiens sont employés à recoder de l'existant, pour rien. Soit parce que la licence du code original est trop restrictive, soit parce que le code original a été volontairement rendu illisible, en mettant des noms de variables aléatoires par exemple (c'est une des techniques d'offuscation). Par ailleurs, les algorithmes comme les formules mathématiques ne peuvent pas être brevetés, mais les procédés techniques qui les utilisent peuvent l'être. Il existe une exception à cette règle : les protocoles de télécommunications [20]. Parfois les aberrations concernent également le matériel, comme les fibres optiques installées par les Fournisseurs d'Acccès à Internet (FAI), dont chaque câble contient en réalité quatre fibres, une par FAI... [21]

Au contraire, certaines associations et entreprises promeuvent les logiciels « open source », dont le code est libre. L'association la plus connue dans ce domaine est la Free Software Fundation, ainsi que sa principale réalisation GNU. En France on retrouve l'April et FramaSoft qui fournit l'outil FramaDate entre autres. Le logiciel VLC fonctionne également grâce à une association. Des entreprises aussi entretiennent le logiciel libre : le code est public, l'utilisation est libre de droits, mais pour demander l'implantation de fonctionnalités spécifiques alors il faut payer. C'est le cas de RedHat (récemment rachetée par IBM), et de Canonical qui conçoivent les systèmes d'exploitations libres fedora et Ubuntu, respectivement. Pour plus d'inspirations, vous pouvez regarder le documentaire La bataille du libre (disponible jusqu'à fin juillet sur Arte en version courte), qui parle par exemple des tracteurs John Deere, irréparables car leur code n'est pas libre.

18. Imaginez un correcteur qui vous met 0 car votre code est incompréhensible, c'est dommage mais ça arrive !
19. Notez que si on se trompe dans les spécifications, là c'est vraiment la merde.
20. Toc toc toc, c'est à l'armée ça, pas touche.
21. Renseignements pris, le non-partage des infrasctures n'est pas imposé par la loi, mais la mutualisation des équipements actifs est soumis à autorisation de l'autorité de la concurrence.